Assignment Guidance and Front Sheet

This front sheet for assignments is designed to contain the brief, the submission instructions, and the actual student submission for any WMG assignment. As a result the sheet is completed by several people over time, and is therefore split up into sections explaining who completes what information and when. Yellow highlighted text indicates examples or further explanation of what is requested, and the highlight and instructions should be removed as you populate 'your' section.

This sheet is only to be used for components of assessment worth more than 3 CATS (e.g. for a 15 credit module, weighted more than 20%; or for a 10 credit module, weighted more than 30%).

To be completed by the student(s) prior to final submission:

Your actual submission should be written at the end of this cover sheet file, or attached with the cover sheet at the front if drafted in a separate file, program or application.

Student ID or IDs for group work	**u2136249**

To be completed (highlighted parts only) by the programme administration after approval and prior to issuing of the assessment; to be consulted by the student(s) so that you know how and when to submit:

Date set	09/12/2022
Submission date (excluding extensions)	10/02/2023
Submission guidance	Via Tabula
Marks return date (excluding extensions)	10/03/2023
Late submission policy	If work is submitted late, penalties will be applied at the rate of **5 marks per University working day** after the due date, up to a **maximum of 10 working days** late. After this period the mark for the work will be reduced to 0 (which is the maximum penalty). "Late" means **after the submission deadline time as well as the date** – work submitted after the given time even on the same day is counted as 1 day late. For **Postgraduate** students only, who started their **current course before 1 August 2019**, the daily penalty is **3 marks** rather than 5.
Resubmission policy	If you fail this assignment or module, please be aware that the University allows students to remedy such failure (within certain limits). Decisions to authorise such resubmissions are made by Exam Boards. Normally these will be issued at specific times of the year, depending on your programme of

	study. More information can be found from your programme office if you are concerned.

To be <u>completed</u> by the <u>module owner/tutor</u> prior to approval and issuing of the assessment; to be <u>consulted</u> by the <u>student(s)</u> so that you understand the assignment brief, its context within the module, and any specific criteria and advice from the tutor:

Module title & code	WM245 Programming Languages for Cyber Security
Module owner	HS Lallie
Module tutor	Dr Nikki Williams
Assessment type	Reflective Report
Weighting of mark	40%

Assessment brief
Please see below

Word count	Described below
Module learning outcomes (numbered)	1. Compare different programming paradigms used to create software. 2. Reflect on how software vulnerabilities can be minimised during software creation. 3. Incorporate security features in small-scale programs. 4. Develop small-scale programs that employ the idioms of a programming paradigm in a conventional manner.
Learning outcomes assessed in this assessment (numbered)	1. Compare different programming paradigms used to create software. 2. Reflect on how software vulnerabilities can be minimised during software creation.
Marking guidelines	Generally indicated within specification
Academic guidance resources	All queries to be directed to the tutor's email, with responses posted via moodle or workshop sessions.

My programming Journey.

As a university student, I have been on a journey of learning to develop my skills, incorporating my interests in cyber security into programming. In this reflective report, I will use Rolfe et al.'s reflective model (Rolfe, 2001) to discuss my programming journey and the steps I have taken to incorporate security into my programming activities. I will start by detailing my programming journey and the steps I have taken to improve my skills and knowledge to give context when focusing on the importance of considering security when developing code. I will finish by discussing my future intentions to consider cybersecurity in all aspects of my programming activities.

My journey in programming has been a combination of self-study and formal education. It began during high school, where I focused on developing a solid foundation in programming concepts such as data types and control structures and following best practices like the "DRY" principle (Muldrow, 2020). Since then, I have continuously learned and developed my skills in low-level, high-level and web application programming languages by taking on more complex projects throughout my academic years.

As I progressed through college, I became more interested in applying programming in cyber security. I was fascinated by using programming languages to think creatively and innovatively to solve complex problems. This interest led me to choose cyber security as my undergraduate degree and has been a driving factor in my efforts to improve my programming skills.

Cybersecurity threats and vulnerabilities are becoming increasingly prevalent in today's digital landscape (Enisa, 2022), which has made me more aware of the importance of considering security when developing any software at any scale. Developers need to consider security early in the software development process. Finding and mitigating potential vulnerabilities before they can be exploited by malicious actors ensures that software and information are secure to protect sensitive information (Snyk, 2021).

During my first year at university, I utilised low-level languages such as C and ASM to gain a concrete understanding of advanced programming concepts and techniques such as pointers, dynamic memory allocation and multi-threading. For example, I developed a multi-threaded network server as part of an assignment that could handle multiple client connections simultaneously. This project involved creating threads for each new client connection, managing the communication between the server and clients, and synchronising access to shared data structures.

Another example is when I worked in assembly language to understand the fundamental architecture of computer systems and the role of the processor in executing instructions. I was tasked to analyse a piece of malware written for the x86 architecture. When analysing the malware, I used reverse engineering techniques. I discovered a SQL injection vulnerability injecting arbitrary code into a website's user input field, which was then executed by the back-end database, allowing the malicious actor to gain sensitive information. To resolve this issue, I disassembled the program to identify the problem. To patch it, I focused on sanitating any user-generated inputs to prevent any malicious SQL code from being executed. This experience enabled me to gain a deeper understanding of memory addressing modes work and how a program interacts with the operating system, gaining insights into how to identify malicious programs and develop more robust applications.

One specific security strategy that has helped me ensure the security of my software applications is input validation. Input validation ensures that user input is in the correct format and is within the expected range of values (Owasp, 2019). By validating user input, it is possible to prevent potential attacks such as SQL injection, which can be used to manipulate a database and allow a threat actor to gain unauthorised access to sensitive information by injecting arbitrary code (OWASP, 2013).

Figure 1 shows a code snippet to illustrate how I have implemented input validation to secure a software application. The following is a Python script that takes user input and searches for a specific record in a database.

```
24  # Validate user input
25  def validate_input(input_string):
26      if len(input_string) > 100:
27          return False
28      if not input_string.isalnum():
29          return False
30      return True
31
32  # Take user input
33  user_input = input("Enter search term: ")
34
35  # Check if input is valid
36  if validate_input(user_input):
37      # Search the database
38      result = search_database(user_input)
39      print(result)
40  else:
41      print("Invalid input. Please try again.")
```

Figure 1 - Implementation of Input Validation

In this code snippet, I have defined a function called “validate_input”, which takes a user input as an argument and checks if it meets specific criteria. First, the function checks if the length of the input string is greater than 100, and if it is, the function returns “False”. This check is in place to prevent potential buffer overflow attacks – a threat actor deliberately tries to input more data in a buffer than it can handle, causing some data to overflow into adjacent storage (Imperva, 2020). The function then checks if the input string contains only alphanumeric characters using the Python built-in method “isalnum()”. If the input string does not meet these criteria, the function returns False. This check prevents potential SQL injection attacks. If the input string meets both requirements, the function returns True, indicating that the input is valid.

In the next part of the code, I have taken user input using the built-in Python function “input()” and passed it to the “validate_input” function. If the input is valid, the code searches the database using the “search_database” function and returns the result. If the input is invalid, the code prints an error message and prompts the user to try again.

By implementing input validation in my code, I can prevent potential attacks and ensure that my software is more secure. This is just one example of how I have integrated my knowledge of cybersecurity into my programming activities, and I intend to continue doing so in the future.

In one of my previous projects, I developed a simple web application that allowed users to upload and share files to further illustrate how I have grown in my programming journey. To ensure user information protection, I used cryptographic practices to process and protect sensitive data such as user’s passwords. First, I sanitised all user input to prevent SQL injection attacks. I then used a secure hashing algorithm to store user passwords instead of storing them in plain text. Additionally, I implemented an access control system to restrict access to sensitive information based on users' roles and permissions to protect users' PII information. To enhance the web application's security, I implemented HTTPS encryption to secure all user data while in transit. This involved obtaining an

SSL/TLS certificate and configuring the web server to use HTTPS by default. To finalise the project, I incorporated a logging and monitoring system to track and detect any unusual user behaviour and prevent potential security breaches.

In the future, I intend to consider cybersecurity in all aspects of my programming activities. This includes incorporating certain coding practices into my code, such as input validation, sanitising user input, and encryption for sensitive data. I also plan to stay updated on cybersecurity threats and vulnerabilities by reading industry publications and attending relevant conferences and workshops. Additionally, I plan to learn more about web and mobile app security, as these areas are becoming increasingly important in today's digital landscape. I intend to do this by familiarising myself with the OWSAP top 10 and OWASP mobile top 10 – these are the top 10 most common security vulnerabilities found in web and mobile applications (OWASP, 2021). I can develop more secure software by understanding these vulnerabilities and how to prevent them.

Furthermore, I plan to incorporate security testing and vulnerability assessments into my software development process. This includes performing penetration testing and code reviews to identify and mitigate potential vulnerabilities in my code. Additionally, I plan to use security tools such as vulnerability scanners and static code analysis to automatically identify and report potential vulnerabilities in my code.

In conclusion, my programming learning journey taught me the importance of cybersecurity when developing software. I now understand the importance of writing secure code and cybersecurity's role in the software development process. By incorporating my cyber security knowledge into my programming activities and using secure coding practices, I can develop more secure software. In the future, I intend to continue considering cyber security in all aspects of my programming activities and stay updated on the latest threats and vulnerabilities in the field.

References

enisa (2022). *Threat Landscape*. [online] ENISA. Available at: https://www.enisa.europa.eu/topics/cyber-threats/threats-and-trends.

imperva (2020). *What is a Buffer Overflow | Attack Types and Prevention Methods | Imperva*. [online] Learning Center. Available at: https://www.imperva.com/learn/application-security/buffer-overflow/.

Muldrow, L. (2020). *What is DRY Development? | DigitalOcean*. [online] www.digitalocean.com. Available at: https://www.digitalocean.com/community/tutorials/what-is-dry-development.

OWASP (2013). *SQL Injection | OWASP*. [online] Owasp. Available at: https://owasp.org/www-community/attacks/SQL_Injection.

OWASP (2021). *OWASP Top Ten*. [online] Owasp.org. Available at: https://owasp.org/www-project-top-ten/.

Owasp (2019). *Input Validation · OWASP Cheat Sheet Series*. [online] Owasp.org. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html.

rolfe (2001). *Rolfe et al's Reflective Framework (2001)*. [online] Available at: https://practicelearning.info/pluginfile.php/317/mod_data/content/3940/Rolfe%20Reflective%20Framework.pdf.

snyk (2021). *Secure Coding Practices | What is secure coding? | Snyk*. [online] snyk.io. Available at: https://snyk.io/learn/secure-coding-practices/.